비휘발성 메모리를 이용하여 데이터 영속성을 유지한 인메모리 키-값 데이터베이스

김도영 ○ 박상현†

연세대학교 컴퓨터과학과

kem2182@yonsei.ac.kr○, sanghyun@yonsei.ac.kr†

A Persistent In-Memory Key-Value Database using Non-volatile memory

Doyoung Kim○ Sanghyun Park†

Department of Computer Science, Yonsei University

요 약

Redis는 메인 메모리에 데이터를 저장하는 인메모리 키-값 데이터베이스로써 실시간 데이터 저장 및 처리가 필요한 서비스에서 많이 사용되고 있다. 하지만, 인메모리 키-값 데이터베이스는 시스템이 종료되면 휘발성을 가지고 있는 메인 메모리의 특성 때문에 저장하고 있는 데이터들이 손실된다. Redis는 데이터 손실을 막기 위하여 메인 메모리의 데이터를 디스크에 저장하는 복구 메커니즘들을 제공한다. Redis에 요청된 명령을 로그로 저장하여 데이터를 디스크에 저장하는 방식인 AOF 복구 메커니즘은 명령어가 요청될 때 마다 로그를 기록하는 always 정책과 매 초마다 로그를 기록하는 everysec 방식으로 구분된다. always 정책은 데이터 영속성을 유지하지만 매번 디스크 연산이 발생하기 때문에 Redis의 성능이 극도로 저하된다. 그에 반해 everysec 정책은 Redis의 성능이 저하되지 않지만, 데이터가 디스크에 동기화가 되지 않고 시스템이 종료되면 데이터 손실이 발생하게 된다. 본 논문에서는 everysec 정책에서 생성된 AOF 로그들을 비 휘발성 메모리에 저장하는 방식을 통해 데이터 손실을 방지하고 데이터 영속성을 유지한 시스템 모델을 제시한다.

1. 서 론

비 휘발성 메모리(Non-volatile Memory)는 전원이 차단되더라도 메모리 안에 데이터가 손실되지 않고 존재하는 특징을 가지는 메모리로서 현재 많이 사용되는 블록 디바이스 스토리지인 하드디스크(HDD) 혹은 솔리드 스테이트 드라이브(SSD)를 대체할 수 있는 디바이스로 각광받고 있다[1]. NVM은 메인 메모리로 사용하는 DRAM에 필적하는 성능을 가지고 있으면서 블록 디바이스처럼 데이터를 영구적으로 보관할 수 있는 특성을 가진다. 인메모리 키-값 데이터베이스는 높은 처리율을 가지지만, 데이터 보존을 위한 복구 메커니즘이 디스크에 데이터를 저장하기 때문에 일시적으로 성능 저하가 발생하게 되거나 데이터 손실이 발생하게 된다. NVM은 DRAM에 필적하는 성능을 보임과 동시에 데이터를 영구적으로 보관할 수 있으므로 디스크를 대신하여 NVM을 활용하는 [2]와 [3] 같은 연구들이 최근에 들어서 많이 진행되고 있다.

※ 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (IITP-2017-0-00477, (SW스타랩) IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발)

† 교신 저자: sanghyun@yonsei.ac.kr

Redis는 상용으로 많이 사용되고 있는 키-값 데이터베이스이다[4]. Redis는 복구 메커니즘으로 append-only file(AOF) 방법을 제공하고 있다[3]. AOF는 요청된 명령어를 메모리에 있는 AOF 버퍼에 임시로 저장했다가 디스크에 로그로 기록하는 방법이다. 이는 디스크 연산을 필요로 하므로 Redis의 성능이 저하되는 요인이 된다. 이를 방지하기 위해 Redis는 백그라운드 쓰레드에서 디스크 I/O를 처리하지만 이 과정에서 데이터 손실이 필연적으로 발생하게 된다. 인텔에서는 NVM을 조작하는 Persistent Memory Development Kit(PMDK) API를 이용하여 Redis의 모든 데이터를 NVM에 저장함으로써 데이터 영속성을 유지하고 성능을 향상시키는 Redis 구현 방식을 제안하였다. 하지만 NVM이 아직 상용화되지 않았기 때문에, 사용할 수 있는 용량에 제한이 있는 단점이 존재한다.

본 논문에서는 AOF 버퍼를 NVM에 구현함으로써 기존의 AOF everysec 정책의 성능을 유지하고 데이터 영속성도 보장하는 Persistent-Buffer Redis (PB-Redis)를 제안한다. 그리고 PB-Redis는 Redis에 저장된 모든 키-값 데이터를 NVM에 유지하지 않고 상대적으로 크기가 작은 AOF 버퍼만 유지함으로써 NVM이 가지는 용량 제한의 문제점을 대처했다.

2. 배 경

2.1 AOF 파일 (Append Only File)

Redis가 받은 명령 중에서 Redis의 데이터를 변경하는 명령들을 AOF파일에 기록하는 방법이다. 명령들은 메모리에서 유지하고 있는 AOF 버퍼에 임시로 저장되었다가 fsync 함수가 호출되면 디스크에 작성된다. fsync 함수 호출 정책에 따라서 Redis의 데이터 영속성이 결정된다. 첫번째로 “always” 정책은 명령이 요청될 때마다 fsync 함수를 호출하는 정책이다. “always” 정책은 데이터 영속성을 유지할 수 있지만 그만큼 디스크 연산이 발생하므로 Redis의 성능이 극도로 저하되게 된다. 그에 반해 “everysec” 정책은 매 초마다 백그라운드 쓰레드에서 fsync 함수를 호출하는 정책이다. “everysec” 정책은 백그라운드 쓰레드에서 디스크 연산이 처리되므로 Redis의 성능을 저하시키지 않지만, fsync 함수가 호출되기 전에 시스템 종료가 발생하면 호출되었던 명령들이 손실되므로 데이터 영속성을 유지할 수 없다.

2.2 PMDK Redis

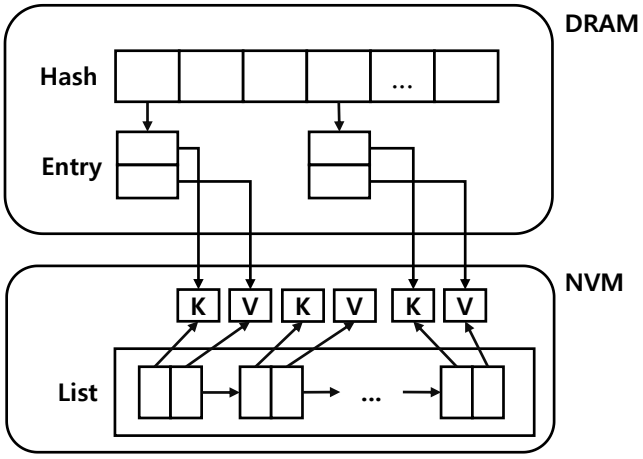

그림 1. PMDK 구현 Redis

인텔에서는 NVM을 조작하는 다양한 트랜잭션 함수들을 인터페이스로 제공하는 PMDK API[4]를 이용하여 Redis의 키-값 데이터를 NVM에 저장하고 유지하는 시스템을 제안했다[5]. 제안한 시스템은 PMDK API를 이용하여 Redis에 저장된 키-값 데이터들을 NVM에 리스트 형태로 유지하고 해쉬 테이블을 DRAM에 구축하여 NVM에 유지된 키-값 데이터에 접근한다. DRAM에 구축되는 해쉬 테이블은 NVM에서 유지하고 있는 리스트에 저장된 키-값 데이터를 이용하여 복구된다.

3. 시스템 설계

본 논문에서는 PMDK 인터페이스를 활용하여 AOF 버퍼를 NVM 에 유지하는 Redis 데이터베이스, PB-Redis 를 제안한다. PB-Redis 의 구현은 다음과 같다. 먼저 데이터베이스를 변경하는 명령이 요청되면 DRAM 에 구축되어 있는 데이터베이스의 데이터를 변경한다. 그 다음, 해당 명령을 NVM 에 리스트로 구현되어 있는 AOF 버퍼에 임시로 저장한다. AOF 버퍼가 임시로 저장되면 백그라운드 쓰레드에서 1 초마다 AOF 버퍼에 저장된 로그들을 디스크에 저장한다. 디스크에 성공적으로 저장되면, 새로운 AOF 버퍼를 생성하고 기존의 AOF 버퍼를 새로 생성한 버퍼로 변경한다. 그리고 기존의 AOF 버퍼를 지우는 작업을 백그라운드 쓰레드에서 수행한다.

명령을 AOF 파일에 쓰기 전까지는 NVM 에 저장된 AOF 버퍼에서 해당 명령 로그 데이터를 유지하고 있기 때문에 시스템이 종료되어도 데이터 영속성이 상실되지 않는다. 그리고 명령 로그 데이터가 디스크의 AOF 파일에 쓰여지고나서 NVM AOF 버퍼의 내용을 지우기 때문에 NVM 이 가지고 있는 제한적인 용량을 무리없이 사용할 수 있다.

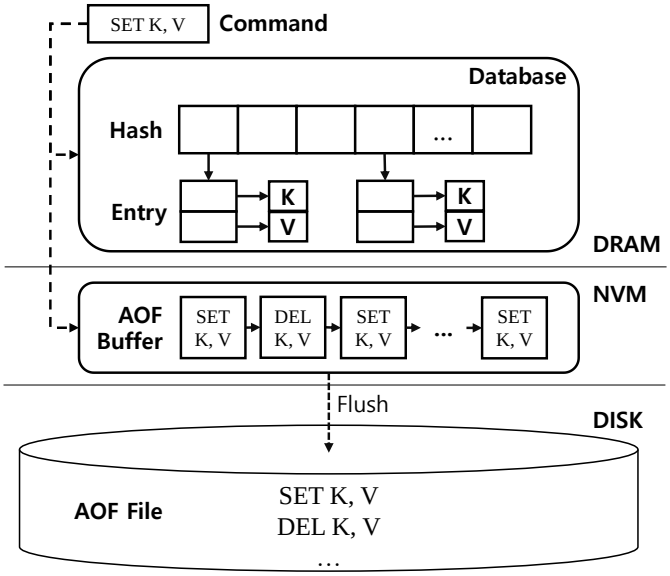

그림 2. PB-Redis의 구조

4. 실 험

표 1 실험 환경

CPU	Intel i7 6700K (Quadcore, 4.0 GHz)
DRAM	64GB (2400MHz)
NVM	8GB (emulated in DRAM)
HDD	3TB (7200RPM)

Redis 에 구현되어 있는 AOF 로그 정책인 “always”와 “everysec”의 성능과 인텔에서 제시한 PMDK 버전 Redis, 그리고 본 논문에서 제시한 시스템의 성능을 측정하고 비교하였다. 첫번째 실험은 Redis 에서 제공하는 벤치마크 프로그램(redis-benchmark)[6]를 사용하여 측정하였다. 그리고 두번째 실험에서는 RedisLabs 에서 구현한 벤치마크

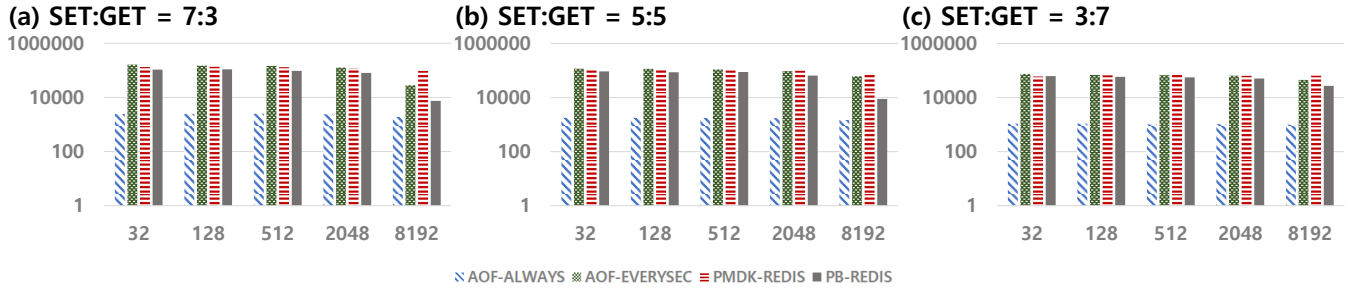

그림 4. Memtier-benchmark 실험 결과

프로그램(memtier-benchmark)[7]를 사용하여 측정하였다. 두 실험 모두 데이터 크기가 32Byte, 128B, 512B, 2048B, 8192B 일때의 시스템 성능을 각각 측정하였다. 실험 환경은 표 1 과 같이 설정하였다. Redis 의 데이터베이스는 DRAM 에 구축되고, AOF 버퍼는 NVM 에 저장된다. 그리고 AOF file 은 HDD 에 저장된다. NVM 은 PMDK 에서 제공하는 NVM 에뮬레이터[8]를 이용하여 DRAM 에 구축하였다.

4.1 Redis-benchmark 실험

Redis-benchmark 프로그램은 Redis 의 간단한 GET, SET 성능을 측정해주는 벤치마크 프로그램이다. 이 실험에서는 AOF-always, AOF-everysec, PMDK-Redis 와 본 논문에서 제시한 시스템인 PB-Redis 의 SET 성능을 측정하였다. 데이터 크기에 따른 실험 케이스에서 요청횟수를 모두 100,000 으로 설정하였다.

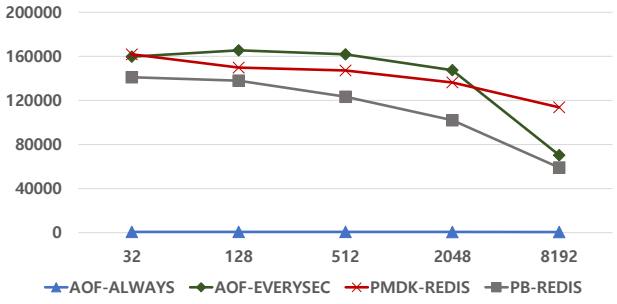

그림 3. Redis-benchmark 실험 결과

실험 결과에서 PB-Redis의 성능이 AOF-everysec과 PMDK-Redis의 성능에 근사하게 측정되는 것을 확인할 수 있었다. 하지만 데이터의 크기가 8192B일 때 성능이 급격하게 감소하였다. PB-Redis는 평균적으로 AOF-everysec보다 1.31배만큼 느리지만 AOF-always보다는 100배 빠른 성능을 보였다. PMDK-Redis와 비교하면, 성능은 PB-Redis가 1.4배 느리지만 NVM의 단점인 제한적인 용량 때문에 가용할 수 있는 NVM 용량에 제약이 존재한다. 반면에, PB-Redis는 상대적으로 더 사용량이 작은 AOF 버퍼만을 NVM에 저장하고 주기적으로 비워지기 때문에 가용할 수 있는 NVM 용량이 더 크다.

4.2 Memtier-benchmark 실험

Memtier-benchmark 프로그램은 인메모리 키-값 데이터베이스의 벤치마크로 자주 활용되는 오픈소스 벤치마크 프로그램이다. Memtier-benchmark 는 GET, SET 이 번갈아 발생하는 상황을 테스트할 수 있는 기능을 제공한다. 4.1 실험과 동일하게 기존의 방법들과 제시한 시스템의 성능을 측정하였다. 그리고 SET 과 GET 의 비율을 다르게 설정하여 측정하였다.

SET 과 GET 이 번갈아 발생하는 상황에서도 PB-Redis 의 성능이 AOF-everysec 과 PMDK-Redis 의 성능에 근사하게 측정되는 것을 확인할 수 있었다. 하지만 4.1 에서 측정된 특징과 동일하게 데이터의 사이즈가 8192B 일 때 성능이 급격하게 감소하였다.

5. 결 론 및 향후 연구

Redis 의 AOF 로그 정책에서 빠른 성능을 가지지만 데이터 영속성을 완벽하게 유지하지 못하는 “everysec”과 데이터 영속성을 유지할 수 있지만 느린 성능을 가지는 “always”의 문제점을 NVM 을 이용하여 데이터 영속성과 빠른 성능을 가지는 AOF 로그 기법을 제안하였다. 그리고 실험을 통하여 대부분의 상황에서 “everysec”에 근접하는 성능을 확인하였다.

하지만 데이터 사이즈가 큰 경우에 AOF 파일을 다시 작성하는 작업이 빈번하게 발생하여 성능이 저하되는 문제점을 확인할 수 있었다. 향후 연구로서 AOF 파일을 다시 작성하는 상황에서도 NVM 에 로그를 유지하는 방식을 적용하여 디스크 파일 접근을 줄여서 성능을 높이는 방식을 구현할 예정이다.

참 고 문 헌

[1] S. Mittal, J. S. Vetter, “A survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems”, *IEEE Transactions on Parrallel and Distributed Systems 27.5*, pp. 1731-1733, May 2016
[2] Y. Son, H. Kang, H.Y. Yeom, H. Han, “A log-structured buffer for database systems using non-volatile memory”, *SAC'17 Proceedings of the Symposium on Applied Computing*, pp. 880-886, Apr 2017
[3] R. Fang, H. Hsiao, B. He, C. Mohan, Y. Wang, “High performance database logging using storage class memory”, *ICDE'11 Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, pp. 1221-1231, Apr 2011
[4] Redis, https://redis.io
[5] Redis Persistence Methods, https://redis.io/topics/persistence
[6] Persistent Memory Development Kit, https://github.com/pmem/pmdk
[7] PMDK implementation Redis, https://github.com/pmem/redis
[8] Redis-benchmark, https://redis.io/topics/benchmarks
[9] Memtier-benchmark, https://github.com/RedisLabs/memtier_benchmark
[10] S. R. Dullor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, J. Jackson, “System software for persistent memory”, *EuroSys'14 Proceedings of the 9th European Conference on Computer Systems*, Article No. 15, Apr 2014